Theoretical Computer Science 21 (1982) 105–111
North-Holland Publishing Company

NOTE

THE MAXIMUM FLOW PROBLEM IS LOG SPACE COMPLETE FOR P

Leslie M. GOLDSCHLAGER*, Ralph A. SHAW and John STAPLES
Department of Computer Science, University of Queensland, St. Lucia, Queensland 4067, Australia

Communicated by J. Ullman
Received May 1981
Revised February 1982

Abstract. The space complexity of the maximum flow problem is investigated. It is shown that the problem is log space complete for deterministic polynomial time. Thus the maximum flow problem probably has no algorithm which needs only $O(\log^k n)$ storage space for any constant k. Another consequence is that there is probably no fast parallel algorithm for the maximum flow problem.

1. Introduction

A long standing open question in the theory of computational complexity is the relationship between time and space bounded computations. For any function $T(n)$, it is not known whether all problems solvable in time $T(n)$ are also solvable in space $O(\log^k T(n))$ for some k independent of n. In particular, it is not known whether all members of P, the class of problems which can be solved in polynomial time, can also be solved in $O(\log^k n)$ space.

This question is particularly interesting in light of the parallel computation thesis [7], which states that the notion of time on an unbounded parallel computer is (polynomially) equivalent to the notion of space. Therefore the question above is equivalent to asking whether or not every computation of a sequential computer has a dramatic speed-up on a parallel machine.

Although the relationship between space and time remains unresolved, Cook [3] exhibited a problem "PATH", the determination of whose space complexity would answer the open question. Cook showed that PATH is among the hardest problems in polynomial time, in the sense that if PATH is in $O(\log^k n)$ space for some constant k, then so is every member of P. PATH is said to be "log space complete for P", and conjectured to require polynomial space. Thus there is

* Present address: Basser Department of Computer Science, University of Sydney, N.S.W. 2006, Australia.

0304-3975/82/0000--0000/$02.75

probably no dramatic speed-up for PATH on a parallel computer. Other problems which are at least as difficult as PATH in this sense are given in [4, 6, 8, 9, 11, 12 and 13].

A natural problem which has received much attention in the literature is the maximum flow problem [5]. A directed graph G with each edge labelled by a non-negative number (called the *capacity* of the edge) and with two distinguished vertices (called the *source* and the *sink*) are given.

A *flow pattern* f is an assignment of a non-negative number to each edge of G (called the *flow* in the edge) such that

(1) there is no edge for which the flow exceeds the capacity and

(2) for every vertex except possibly the source and the sink, the sum of the flows on its incoming edges equals the sum of the flows on its outgoing edges (i.e. there is conservation of flow).

The *maximum flow problem* is to compute a flow pattern f whose total flow F into the sink is maximum.

In this paper, we show that the maximum flow problem is at least as difficult as PATH. In fact the computation of the value F of the maximum flow (and indeed the computation of just the least significant bit of F) is at least as difficult as PATH in the sense mentioned above.

This result implies that if the maximum flow problem can be solved in $O(\log^k n)$ space (parallel time) for some constant k, then every problem which can be solved in polynomial time could also be solved in $O(\log^k n)$ space (respectively parallel time). Thus the result of this paper can be regarded as some evidence that no dramatic speed-up is possible for the maximum flow problem on a parallel machine.

2. Definitions

We use standard definitions of space and time, see for example [10 and 11]. Let P be the class of languages recognizable in deterministic polynomial time, and let A and B be arbitrary languages.

Definition. $A \leqslant_{\log} B$ (read "A is *log space transformable* to B") iff there exists a function f which can be computed in log space and, for any w, $w \in A$ iff $f(w) \in B$.

Definition. B is *log space complete for* P iff $B \in \mathrm{P}$ and for every $A \in \mathrm{P}$, $A \leqslant_{\log} B$.

Lemma. *If B is log space complete for* P *and $B \leqslant_{\log} A$ and $A \in \mathrm{P}$, then A is log space complete for* P.

Lemma. *If B is log space complete for* P *and B is recognizable in space $O(\log^k n)$ for some constant $k \geqslant 1$, then, for every $A \in \mathrm{P}$, A can be recognized in space $O(\log^k n)$.*

Definition. A *monotone circuit* α is a sequence $(\alpha_n, \ldots, \alpha_0)$ where each α_i is either an input or a gate AND(j, k) or OR(j, k) where $j \geq k > i$ and the values of the inputs are given explicitly. The *monotone circuit value problem* MCVP $= \{\alpha \mid$ the output gate α_0 computes true$\}$.

Intuitively, MCVP is the problem of determining the value computed by the output gate α_0 of an explicitly presented combinational circuit α, given the values of its inputs. α consists only of AND and OR gates.

Lemma ([8]). MCVP *is log space complete for* P.

Definition. Let β be a monotone circuit $(\beta_n, \ldots, \beta_0)$ such that

(1) if β_i is an input then the index i appears at most once in β (i.e. each input has fan-out at most one),

(2) if β_i is a gate, then i appears at most twice in β (i.e. each gate has fan-out at most two) and

(3) β_0 is an OR gate with fanout 1.

MCV2 $= \{\beta \mid$ the output gate β_0 computes true$\}$.

Lemma. MCV2 *is log space complete for* P.

Sketch of proof. Clearly MCV2 $\in$ P and the construction below shows that MCVP $\leq_{\log}$ MCV2.

Given a monotone circuit $\alpha = (\alpha_m, \ldots, \alpha_0)$, the basic idea is to construct a monotone circuit β with the required properties ((1)–(3) above) by replacing each gate α_i by a fan-out tree with M leaves (where M is the smallest power of two so that $M > m$). The fan-out tree corresponding to α_i will be gates $\beta_{(i+1)(2M-1)-1}$, $\beta_{(i+1)(2M-1)-2}, \ldots, \beta_{i(2M-1)}$. More formally, if α_i is a gate with inputs j and k, let $j' = j(2M-1)+i$ and $k' = k(2M-1)+i$ and let $\beta_{(i+1)(2M-1)-1}$ be AND(j', k') or OR(j', k') depending upon whether α_i is an AND or an OR gate respectively. Let the gates $\beta_{(i+1)(2M-1)-2}, \ldots, \beta_{i(2M-1)}$ be a binary fan-out tree with M leaves whose internal nodes are OR gates (using some dummy input which will be given the value zero). If α_i is an input let each corresponding β be the same input. Intuitively, the gate $\beta_{(i+1)(2M-1)-1}$ simulates the gate α_i but takes its inputs j' and k' from unique leaves of the fan-out tree for α_j and α_k. We note that properties (1)–(3) hold for β and we claim that the construction can be performed by a log space bounded Turing machine. □

Definition. Let G be a directed graph with non-negative edge labels. The problem of *computing the value of the maximum flow* is

MAXFLOW $= \{\langle G, s, t, i\rangle \mid i$th bit of the value of the maximum flow from source s to sink t in G is 1$\}$.

3. Main result

In this section we prove that MCV2 is log space transformable to MAXFLOW, and hence that MAXFLOW is log space complete for P.

Theorem. MAXFLOW *is log space complete for* P.

Proof. MAXFLOW $\in$ P [5] and the construction below shows that MCV2 $\leqslant_{\log}$ MAXFLOW.

Let $\beta = (\beta_n \cdots \beta_0)$ be an arbitrary monotone circuit with properties (1)–(3) above. We construct an edge labelled directed graph G as follows.

The vertices V of the graph are s and t (source and sink respectively) and i for all $0 \leqslant i \leqslant n$.

The edges E of the graph are for all $0 \leqslant i \leqslant n$,

(a) If β_i is an input then (s, i) with capacity 0 if β_i is false and capacity 2^i if β_i is true. Also $(i, s) \in E$ with capacity 2^i.

(b) If $\beta_i = \text{AND}(j, k)$ then

$$(j, i) \in E \text{ with capacity } 2^j,$$

$$(k, i) \in E \text{ with capacity } 2^k$$

and

$$(i, t) \in E \text{ with capacity } 2^j + 2^k - d2^i$$

where d is the fan-out of β_i (recall that $d = 0$, 1 or 2).

(c) If $\beta_i = \text{OR}(j, k)$ then

$$(j, i) \in E \text{ with capacity } 2^j,$$

$$(k, i) \in E \text{ with capacity } 2^k$$

and

$$(i, s) \in E \text{ with capacity } 2^j + 2^k - d2^i$$

where d is the fan-out of β_i.

Finally, $(0, t) \in E$ with capacity 1. Fig. 1 gives an illustration of the construction.

We claim that the value of the maximum flow in G is odd iff the circuit outputs true. That is $\beta \in$ MCV2 iff $\langle G, s, t, 0\rangle \in$ MAXFLOW. The theorem follows from this claim together with the observation that the construction can be performed by a log space bounded Turing machine. It remains to prove the claim.

Definition. The *simulating flow pattern* f_s is defined as follows.

For all $0 \leqslant i \leqslant n$,

(a) If β_i is an input then the flow in (s, i) equals the capacity. Also if the index i does not appear as the input to any gate, then the flow in (i, s) is equal to the flow in (s, i), otherwise it is zero.

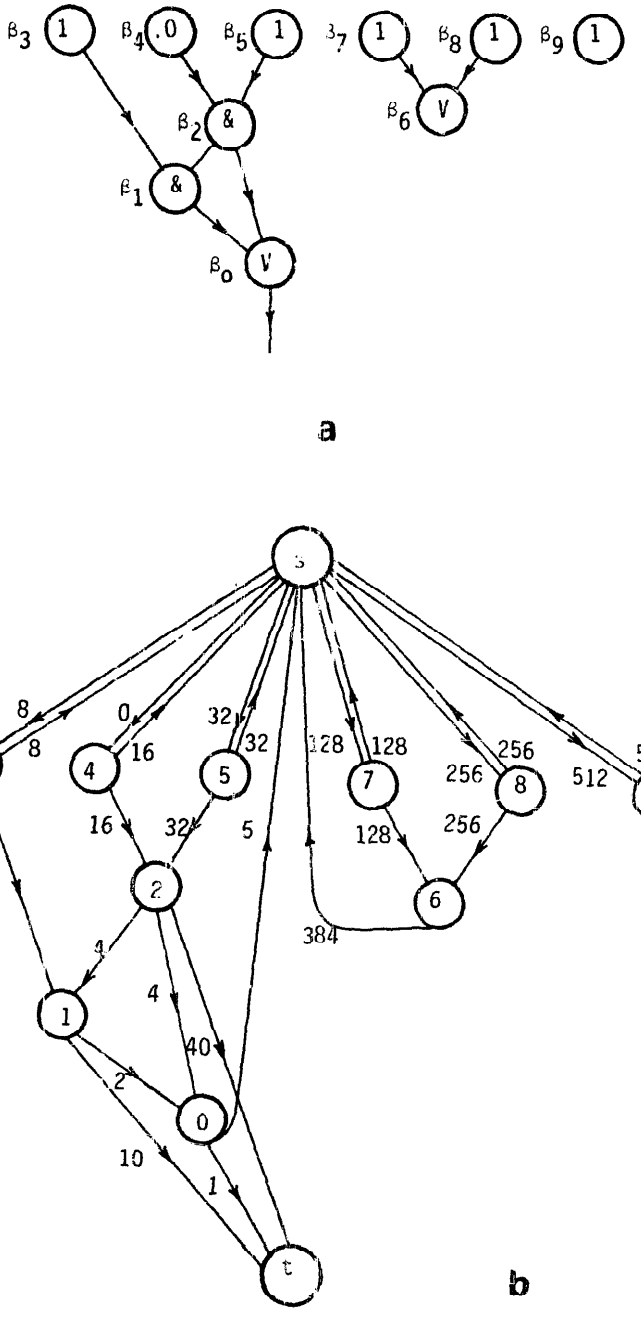


Fig. 1. (a) A circuit β; (b) Its corresponding graph G.

(b) For all $0 \leq j \leq n$ the flow in (i, j) is 2^i if β_i computes true and 0 otherwise.

(c) If $\beta_i = \text{AND}(j, k)$ then the flow in (i, t) is equal to the capacity if both β_j and β_k compute true. Otherwise the flow in (i, t) equals the sum of the flows of (j, i) and (k, i).

(d) If $\beta_i = \text{OR}(j, k)$ then the flow in (i, s) is equal to the sum of the flows of (j, i) and (k, i) minus $d2^i$ if either β_j or β_k computes true. Otherwise the flow in (i, s) is zero. (Recall that d is the fan-out of β_i).

Finally, the flow in $(0, t)$ is 1 if β_0 computes true and zero otherwise.

It is easy to check that f_s is indeed a flow pattern (using properties (1) and (2) of β). That is, no edge is assigned a flow greater than its capacity, and there is conservation of flow at every vertex (except possibly s and t).

Definition. An *augmenting path* is a sequence of distinct vertices beginning with the source and ending with the sink such that if i and j are successive vertices in the sequence then either

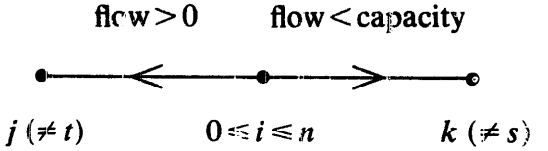
(a) G contains an edge (i, j) whose flow is less than its capacity (called a "*forward edge*"), or

(b) G contains an edge (j, i) whose flow exceeds zero (called a "*back edge*").

Lemma ([5]). *A flow pattern has an augmenting path iff its total flow is less than the maximum possible flow.*

Lemma. *The simulating flow pattern f_s achieves the maximum possible flow in G.*

Proof of lemma. Any augmenting path in f_s must begin with a back edge since each forward edge out of s has its flow equal to its capacity, and must end with a forward edge since there are no edges out of t. Thus there must be a back edge followed by a forward edge somewhere along the path:



But by the construction of f_s,

(a) β_i is not an input, otherwise $j = s$ and the flow in (i, j) would be zero;

(b) β_i is not an AND gate, otherwise the fact that the flow in $(i, j) > 0$ would imply that the flow in (i, k) equals its capacity; and

(c) β_i is not an OR gate, otherwise the fact that the flow in (i, k) is less than its capacity would imply that the flow in (i, j) is zero.

Thus there is no augmenting path in f_s. □

Note from the construction of f_s that f_s assigns an even valued flow to every edge (i, t) except possibly the edge $(0, t)$, (using property (3) of β). Thus the total flow of f_s is odd iff f_s assigns flow 1 to edge $(0, t)$. That is, the maximum flow in G is odd iff the output gate of the circuit computes true. We have proved that $\beta \in$ MCV2 iff $\langle G, s, t, 0\rangle \in$ MAXFLOW, as required. □

4. Conclusion

We have shown that MAXFLOW is log space complete for P, thereby providing some evidence that there is neither an $O(\log^k n)$ storage algorithm nor an $O(\log^k n)$ parallel time algorithm for computing the maximum flow, for any constant k.

Here are some related open problems. Do similar results hold concerning the maximum flow in graphs with edge capacities substantially less than 2^n? In particular, the problems of polynomial capacities and 0–1 capacities appear interesting [5]. One special case where there is an $O(\log^2 n)$ space algorithm for computing the value of the maximum flow is planar graphs which can be embedded so that s and t appear on a common face. This follows from an observation made by Dan Johnston that the maximum flow problem can be expressed as a path problem in the dual graph.

A further open problem is to determine the space complexity of POSSIBLEFLOW $= \{\langle G, s, t, F\rangle \mid F \leq$ the maximum flow in G from s to $t\}$.

References [1] and [2] discuss the problem of finding the maximum flow through a *single* path from source to sink, and should not be confused with the classical maximum flow problem studied in this paper.

Acknowledgment

We thank Larry Ruzzo and Neil Jones for their simplifications to the lemma in Section 3.

References

[1] I. Chen, A new parallel algorithm for network flow problems, Lecture Notes in Computer Science **24** (Springer, Berlin, 1975) 306–307.

[2] Y.K. Chen and T. Feng, A parallel algorithm for the maximum flow problem, *Proc. 2nd Sagamore Computer Conference on Parallel Processing*, Raquette Lake, NY (1973) 60.

[3] S.A. Cook, An observation on time-storage trade off, *J. Comput. System Sci.* **9** (3) (1974) 308–316.

[4] D. Dobkin, R.J. Lipton and S. Reiss, Linear programming is log-space hard for P, *Information Processing Lett.* **9** (2) (1979) 96–97.

[5] S. Even, *Graph Algorithms* (Pitman, London, 1979).

[6] Z. Galil, Hierarchies of complete problems, *Acta Informat.* **6** (1976) 77–88

[7] L.M. Goldschlager, A unified approach to models of synchronous parallel machines, *Proc. 10th Annual ACM Symposium on Theory of Computing* (1978) 89–94.

[8] L.M. Goldschlager, The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9** (2) (1977) 25–29.

[9] L.M. Goldschlager, ϵ-productions in context-free grammars, *Acta Informat.* **16** (1981) 303–308.

[10] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).

[11] N.D. Jones and W.T. Laaser, Complete problems for deterministic polynomial time, *Theoret. Comput. Sci.* **3** (1) (1976) 105–117.

[12] D. Kozen, Complexity of finitely presented algebras, *Proc. 9th Annual ACM Symposium on Theory of Computing* (1977) 164–177.

[13] R.E. Ladner, The circuit value problem is log space complete for P, *SIGACT News* **7** (1) (1975) 18–20.